Wiki Semantique

TO52 – Travail Opérationnel

Présentation du projet de TO52 – Wiki Sémantique. Conception, développement et Manuel d'installation.

CONIGLIO Christophe
VINCENT Jérémy

23/01/12

SUIVI DU DOCUMENT

Versions du document

Version	Modifications	Date	Pages concernées
1.0.0	Création du document	19/12/2011	Toutes
1.0.1	Rédaction	19/12/2011	Toutes
1.0.2	Modification du document	29/12/2011	Toutes
1.0.3	Modification du document	07/01/2012	Toutes
1.1.0	Modification du document	23/01/2012	Toutes
1.1.1			

Auteurs

Nom	Prénom	Département	Versions
VINCENT	Jérémy	GI05 – ILC	1.0
CONIGLIO	Christophe	GI05 – I2RV	1.0

Destinataires

Nom	Adresse mail	Enseignement	Versions
HILAIRE Vincent	vincent.hilaire@utbm.fr	TO52	1.0
LAHOUD Inaya	inaya.lahoud@utbm.fr	TO52	1.0

Approbation du document

Etape	Nom	Signature
Rédaction	CONIGLIO Christophe VINCENT Jérémy	
Validation	HILAIRE Vincent LAHOUD Inaya	

Introduction

Dans le cadre de l'UV TO52, nous avions à notre charge la réalisation d'un projet personnel. A ce titre, nous devions créer un wiki sémantique, permettant ainsi la gestion de la sémantique et des ontologies entre les articles.

Basé sur une technologie de type java/JEE + RDF, ce projet est facilement intégrable à un projet à plus grande échelle.

Au sein de ce rapport les différentes informations relatives à nos recherches ainsi que les différents outils et méthodes utilisées, en passant par un manuel d'installation et d'utilisation de l'application seront détaillés.

Table des matières

Suivi du document 1

Introduction 2

I. Descriptif du sujet 4
- a. Sujet 4
- b. Pistes explorées 5
- c. Contraintes d'utilisation 7
- d. Spécifications techniques 6

II. Développement du wiki 9
- a. Maquettes 9
- b. Architecture de l'application 12
- c. Fonction de recherche – Jena 15

III. Evolutions possibles 19
- a. Evolutivité de la fonction recherche 19
- b. Ajout de nouvelles fonctionnalités 20
  - ✓ Création manuelle de nouveaux articles 20
  - ✓ Gestion des utilisateurs 20
  - ✓ Parseur RDF 20

Conclusion 21

Annexes 22
- Exemple d'un ficher RDF 22
- Manuel d'installation 23

I. DESCRIPTIF DU SUJET

a. Sujet

Dans le cadre de cette UV, nous devions donc mettre en place ce système de wiki sémantique. Ce projet a pour but de réaliser un système de données de structure sémantique et de réaliser une interface graphique permettent de d'exploiter ces données, d'effectuer des recherches au sein du système et également d'y effectuer des modification.

Les données fournies via des fichiers RDF (dont nous expliquons les spécificités par la suite) apportent une meilleure gestion de la sémantique entre les articles, le but de ce projet étant en effet d'interpréter la signification ainsi que le sens des articles (leur sémantique) afin de les regrouper par concepts représentant justement leur sémantique (regroupés par ontologie).

b. Définitions

- **Wiki** : Site web dont les pages sont modifiables par les visiteurs afin de permettre l'écriture et l'illustration collaboratives des documents numériques qu'il contient.

- **Sémantique** : La sémantique est une branche de la linguistique qui étudie les signifiés. A rapport à la signification, au sens.

- **Ontologie** : Ensemble structuré des termes et concepts représentant le sens d'un champ d'informations, que ce soit par les métadonnées d'un espace de noms, ou les éléments d'un domaine de connaissances.

- **Syntaxe RDF** : Modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions. RDF est le langage de base du Web sémantique. La syntaxe de ce langage est basée sur XML. L'API Jena pour JAVA permet le traitement de telles fichiers.

c. Pistes explorées

Lors de la phase de recherche de pistes à explorer et de collecte d'informations, nous avons eu à faire face à différents types de technologie. Nous allons vous présenter ici les différentes pistes explorées et les choix qui se sont avérés être les meilleurs pour nous.

JSPWiki

JSPWiki est la première piste sur laquelle nous avons effectué des recherches. Basée sur un moteur Java/JEE, cette application permet de générer un système de wiki complet. Séduits dans un premier temps, et après plusieurs recherches, nous avons fini par conclure que ce projet était en perte totale de vitesse, et ne nous permettait pas de l'intégrer dans une logique dynamique. De plus, le code source étant accessible sans aucune documentation et annotation quelconque, il nous aurait été très difficile d'adapter notre code afin de la rendre compatible avec nos attentes.

MediaWiki

MediaWiki est un Framework développé en PHP/MySQL permettant de générer les bases d'un wiki prêt à l'emploi. Très simple d'utilisation et relativement performant, il est alors devenu intéressant de travailler avec.

Après l'avoir installé et testé, nous avons trouvé différents modules tels que la notation d'articles et la gestion sémantique des articles. L'installation est facile est le Wiki installé fonctionne parfaitement, mais la base de donnée MySql est très complexe et il y est donc difficile d'effectuer des modifications sur la recherche. De plus, il nous aurait fallu intervenir dans le code source de l'application car il était indispensable pour nous de gérer les données au travers les fichiers RDF et non à l'aide d'une base MySQL. Nous avons donc convenu d'explorer d'autres pistes.

L'API Jena

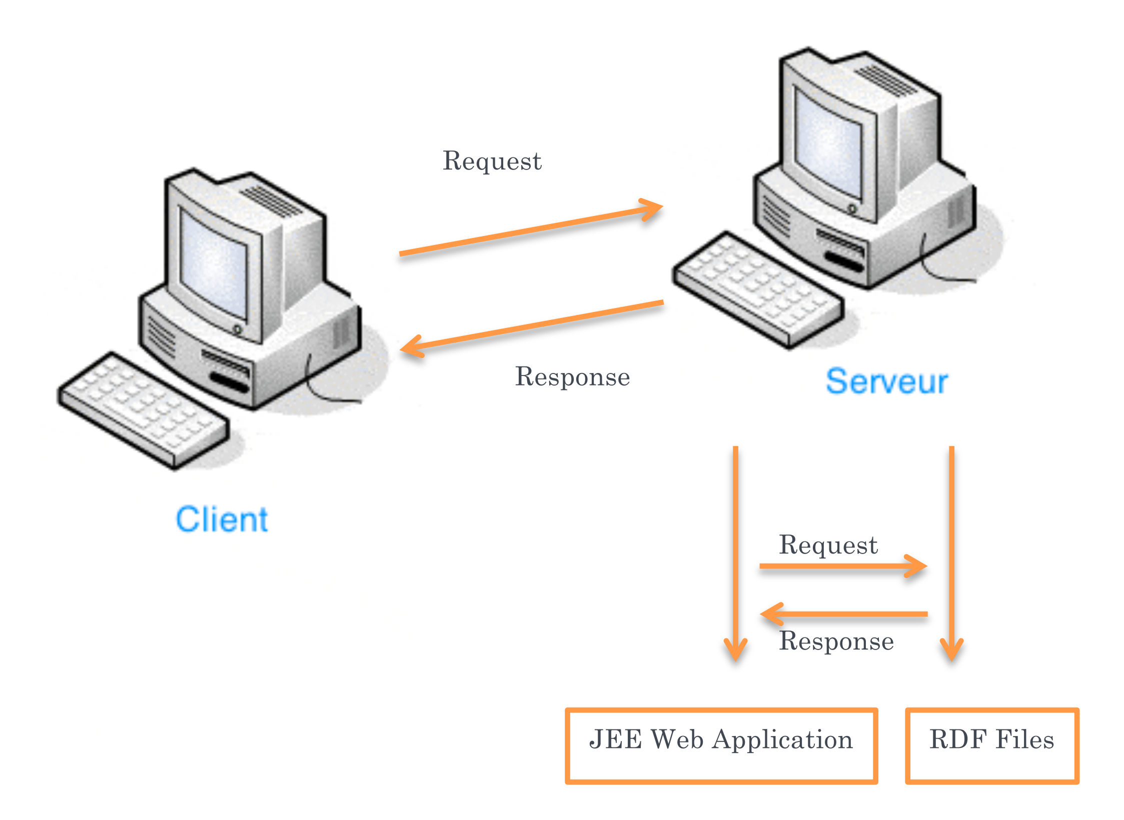
L'API Jena est une bibliothèque de classes permettent de manipuler des fichiers RDF et d'y effectuer des recherches en son sein. Nous avions en notre possession différents exemples issus de la documentation de Jena permettant de manipuler des systèmes de données appelés « Model » et nous étions en mesure d'exécuter de simples exemples de recherche avec des requête ARQL réalisées sans grande difficulté grâce à une bonne documentation. Le principal inconvénient de Jena est qu'il est programmé sur Java ce qui nécessite un serveur JEE pour la réalisation de projet Web avec Jena.

C'est alors que nous avons convenu de reprendre le projet "from scratch" et donc de partir de rien afin d'implémenter notre propre système.

JEE/JSP

Afin de faire face aux différentes contraintes de développement, et afin de rendre l'intégration du projet plus simple, nous avons décidé de développer nous même notre propre wiki basé sur un moteur java/JEE. Ce wiki reste très basique mais offre les fonctionnalités principales telles que la visualisation, modification, notation d'articles. De plus, nous avons à l'aide de Jena, intégré de manière plus propre le système de gestion sémantique des articles.

d. Spécifications techniques



Pour fonctionner, la système dispose d'une base de fichiers RDF, servant de base de données à notre application. Pour ce faire, lorsque le client envoie une requête à notre serveur, le serveur Tomcat interprète cette requête et la retransmet à notre application qui va alors effectuer des traitements au sein des fichiers RDF afin de retourner les données désirées.

Au sein de notre application, nous disposons d'une classe "JenaInit" permettant d'effectuer toutes les opérations (notamment de recherche), propres à Jena et à l'exploitation des fichiers RDF.

Une classe Article permettant de gérer dans sons intégralité les articles du wiki.

Notre wiki utilise le modèle MVC (modèle, vue, contrôleur).

Les vues étant principalement des pages JSP, implémentées à l'aide du HTML, de JSP et parfois la JSTL de Sun. (librairie de tags évitant au développeur d'écrire des scriptlets dans des pages JSP.

Les contrôleurs étant quand-à eux les servlets créés afin de vérifier l'intégrité des données et d'exploiter les données fournies.

Les modèles étant les classes de base listées ci-dessus, permettant la gestion des objets.

Pour résumer, les servlet créent les modèles qui sont à leur tour affichés (préalablement exploités) dans les vues.

e. Contraintes d'utilisation

Le fichier listeRDF

Au sein de notre application, nous disposons un fichier XML récapitulant l'adresse de tous nos fichiers RDF sur le serveur. Ainsi, il est plus facile d'exploiter les données. Afin de ne pas à modifier directement le code Java pour chaque changement sur la base de données créée à partir des fichiers RDF, nous avons créé un fichier listeRDF qui regroupe la liste de tous les fichier RDF ce qui permet de le modifier à l'aide d'un simple éditeur de texte pour un utilisateur averti. En effet, pour palier à ce problème, une fonction interne à l'application nommée servant à mettre à jour ce fichier accessible via l'appel à la fonction d'upload de nouveaux fichiers dans le menu permet de régénérer ce fichier automatiquement.

Contrainte sur les fichiers RDF

Nous avons choisi d'établir que chaque donnée aura son propre fichier RDF. Pour chaque fichier nous ajoutons donc les nœuds suivants :

- « NameRDF » qui correspond au nom du fichier RDF ce qui permet de trouver son adresse sur le serveur.

- « Parent » qui correspond au nom du fichier parent du fichiers RDF ce qui permet de gérer l'héritage des fichier RDF.

- « Rate » & « nbRaters » qui permettent de noter la pertinence de chaque article enregistré dans la base. En effet, la note de l'article est obtenue en divisant la note globale (rate) par le nombre de votants (nbRaters).

Informations Importantes

Selon l'ontologie de l'article en question, le fichier possède des balises identiques quelle que soit cette ontologie (NameRDF, RoleCreateurConnaissance, CrerPar, DateDebut, DateFin, IndiceMaturite, PourcentageEvaluation, Commentaire, Rate, NbRaters, Parent). Celles-ci se devront d'être présentes dans chaque fichier. En cas d'absence, une erreur s'ensuivra. D'autres balises peuvent venir s'ajouter afin de compléter la source informative, et viendront alors s'ajouter automatiquement en tant qu'informations supplémentaires dans l'interface de l'application. (Cf. annexe : exemple de fichier RDF).

Il est crucial, lors de la création de nouveaux fichiers RDF, pour alimenter la base de données, de respecter **scrupuleusement** la notation XML/RDF. En effet, une balise mal formée, une casse non respectée ou autre entrave à la bonne formation d'un fichier RDF entrainerait un disfonctionnement de l'application.

Description et utilisation d'un fichier RDF

```
<rdf:RDF
xmlns:CyclingOnto="http://KATRAS/AcivitePropriete/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
--Syntaxe de base de la même façon qu'un fichier XML aurait la syntaxe (< ?xml version…)
<rdf:Description rdf:about="http://utbm.acsp.fr/CP51_LO15_SlowMax Tri One/">
--Ligne récapitulant ce quoi parle le fichier RDF avec la propriété about égale à l'url
http://utbm.acsp.fr/ suivie du nom de fichier
<CyclingOnto:NameRdf>SlowMax-TriOne</CyclingOnto:NameRdf> (2)
--Le nom du fichier RDF sans l'extension. Fait office d'identifiant
<CyclingOnto:RoleCreateurConnaissance>Concepteur</CyclingOnto:RoleCreateurConnais
sance>
--Le rôle du fichier, en gros sa catégorie (voir en fonction des besoins)
<CyclingOnto:CreerPar>ILahoud</CyclingOnto:CreerPar>
--Auteur du fichier
<CyclingOnto:DateDebut>08/01/2009</CyclingOnto:DateDebut>
<CyclingOnto:DateFin>05/01/2012</CyclingOnto:DateFin>
--Dates de creation et de dernière modification
<CyclingOnto:Commentaire> SlowMax CF 2010 TriOne</CyclingOnto:Commentaire>
--Contenu textuel de la donnée
<CyclingOnto:Rate>12</CyclingOnto:Rate>
<CyclingOnto:NbRaters>3</CyclingOnto:NbRaters>
--Note globale / nombre de votants = note sur 5 de l'article.
<CyclingOnto:Parent>SlowMax</CyclingOnto:Parent>
--Article Parent (héritage)
<CyclingOnto:FrameWeight>1.700</CyclingOnto:FrameWeight>
<CyclingOnto:FramesetLength>XL</CyclingOnto:FramesetLength>
<CyclingOnto:SeatTubeLength>3600</CyclingOnto:SeatTubeLength>
<CyclingOnto:TopTubeLength>585</CyclingOnto:TopTubeLength>
<CyclingOnto:HeadTubeLength>175</CyclingOnto:HeadTubeLength>
<CyclingOnto:HeadTubeAngle>72</CyclingOnto:HeadTubeAngle>
<CyclingOnto:SeatTubeAngle>75</CyclingOnto:SeatTubeAngle>
<CyclingOnto:ChainstayLength>3900</CyclingOnto:ChainstayLength>
<CyclingOnto:WheelBase>1010</CyclingOnto:WheelBase>
<CyclingOnto:HandlebarName>Alu X</CyclingOnto:HandlebarName>
<CyclingOnto:HandlebarWeight>927</CyclingOnto:HandlebarWeight>
<CyclingOnto:HandlebarMaterial>Aluminuim</CyclingOnto:HandlebarMaterial>
<CyclingOnto:HandlebarLength>36</CyclingOnto:HandlebarLength>
<CyclingOnto:WheelName>Zipp 808 AR</CyclingOnto:WheelName>
<CyclingOnto:WheelWeight>808</CyclingOnto:WheelWeight>
<CyclingOnto:WheelMaterial>Carbone</CyclingOnto:WheelMaterial>
<CyclingOnto:WheelSize>26</CyclingOnto:WheelSize>
<CyclingOnto:cranksetName>Speed-One</CyclingOnto:cranksetName>
<CyclingOnto:cranksetWeight>112</CyclingOnto:cranksetWeight>
<CyclingOnto:cranksetMaterial>Alliage Alu-Carbone</CyclingOnto:cranksetMaterial>
<CyclingOnto:ChainRingTeeth>54</CyclingOnto:ChainRingTeeth>
--En orange, les données facultatives traitées à part. Entièrement modifiable en fo,ction de
l'ontologie. En effet, ces données ne seront pas les memes pour deux ontologies différentes
(par exemple vélo et cuisine)
</rdf:Description>
</rdf:RDF>
--Fermeture des balises de base, ainsi respect de la syntaxe XML/RDF
```

II. Developpement du wiki

a. Maquettes

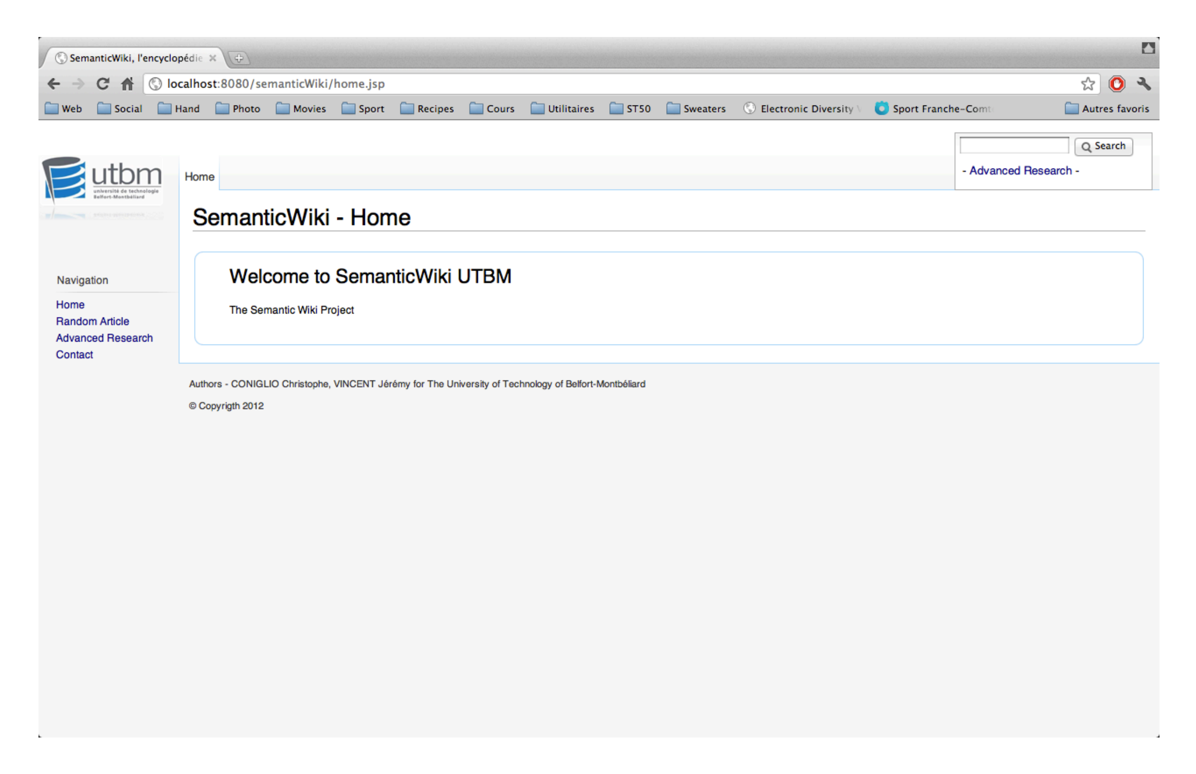


Figure 1 : Page d'accueil

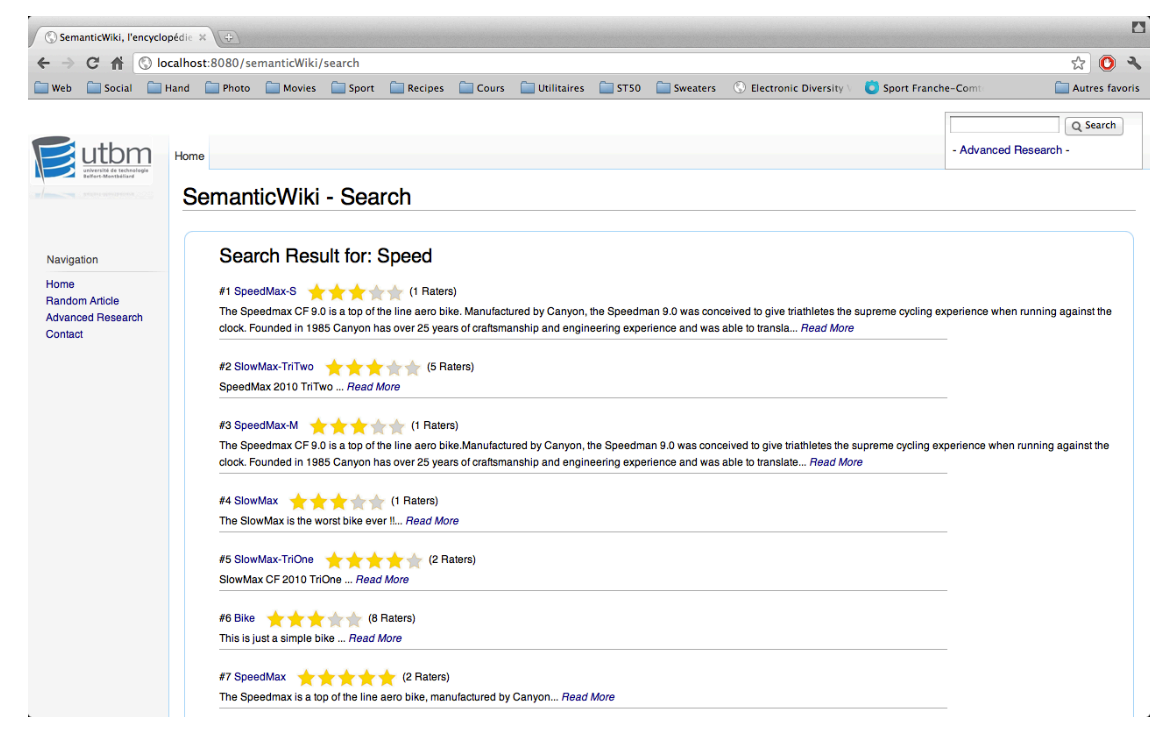


Figure 2 : Fonction de recherche simple

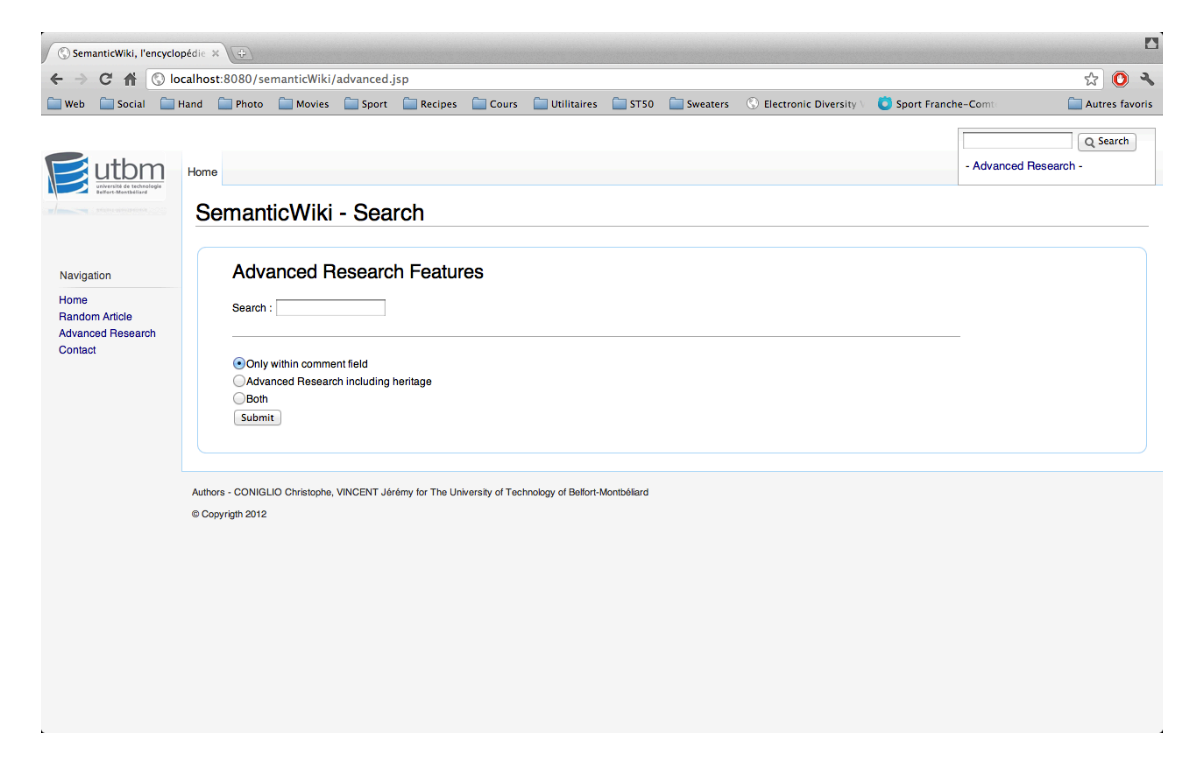


Figure 3 : Fonction de recherche avancée

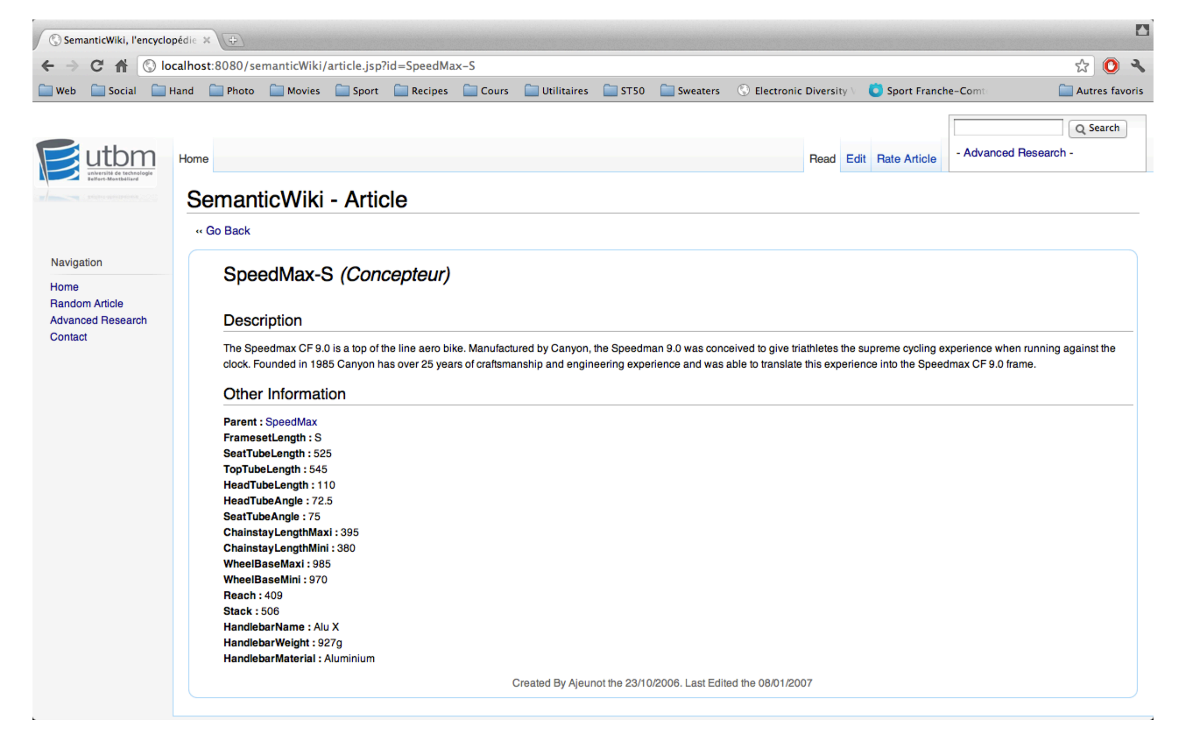


Figure 4 : Afficher Article

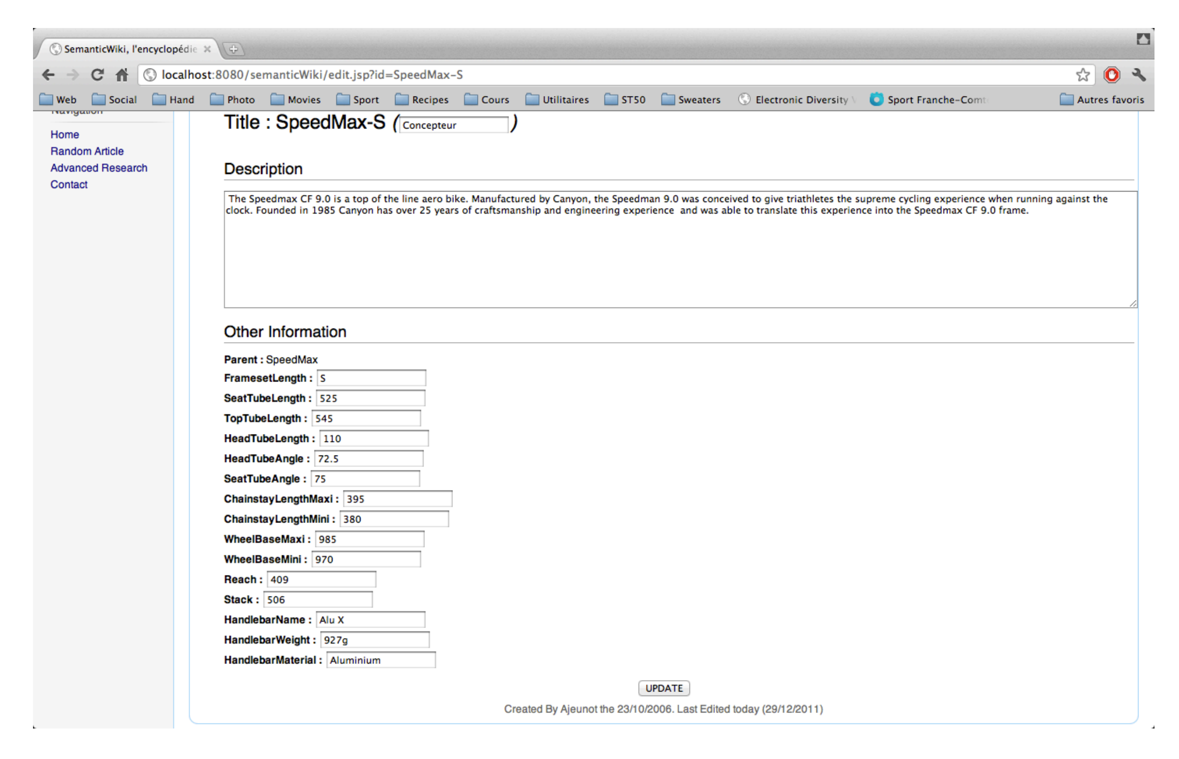


Figure 5 : Modifier Article

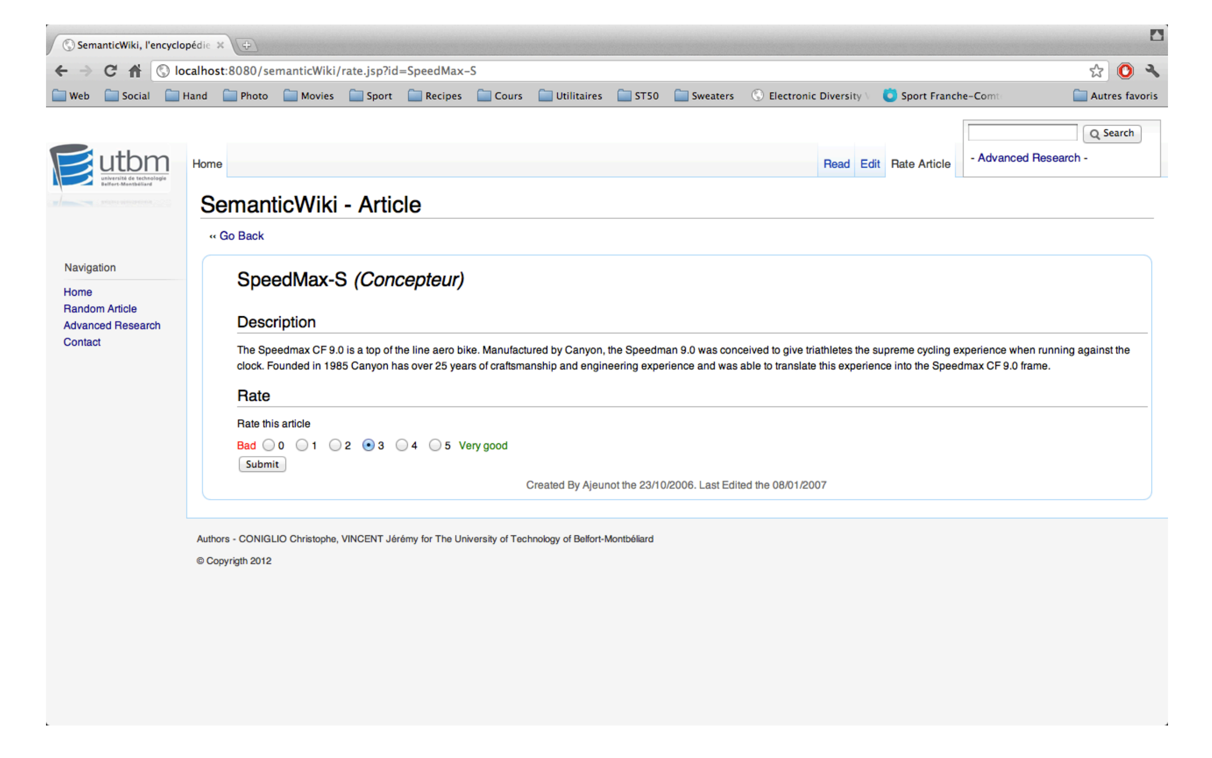


Figure 6 : Noter Article

b. Architecture de l’application

L’application est divisée en trois parties principales. Les modèles, les vues et les contrôleurs.

Modèles

Les modèles sont les deux classes listées dans les spécifications techniques, à savoir JenaInit et Article.

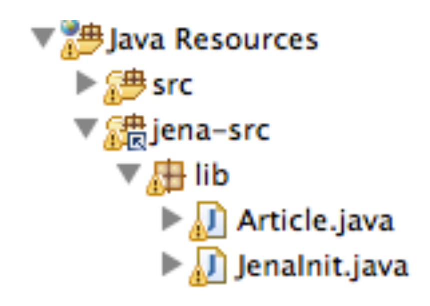
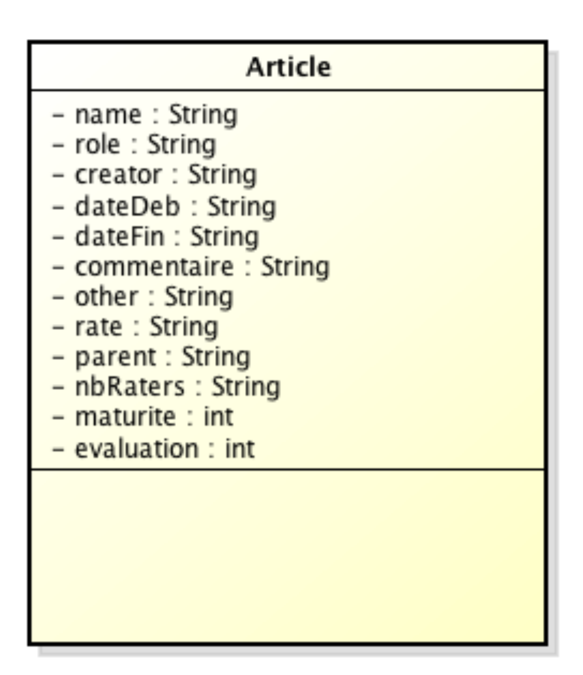


Figure 7 : Liste des classes de base - Models



JenaInit
- Static model : Model
- static arraylistNameRdf : ArrayList<String>
- static arraylistParent : ArrayList<String>

Figure 8 : Classe JenaInit

Figure 9 : Classe Article

La classe Article permet toutes les modifications relatives aux données de la base. En effet, grâce à cette classe, il est possible d’accéder aux données d’un fichier RDF, ainsi que de les modifier.

La classe JenaInit permet quand-à elle d’effectuer toute opération de recherche dans les fichiers. En effet, elle a pour fonction principale de rechercher les articles concernés par la recherche saisie et de retourner un résultat prenant en compte l’héritage entre les différents articles.

Vues

Ci-dessous, le contenu du dossier WebContent de l'application. Dans ce dossier, les différents fichiers tels que les vues JSP ainsi que les sources de données.

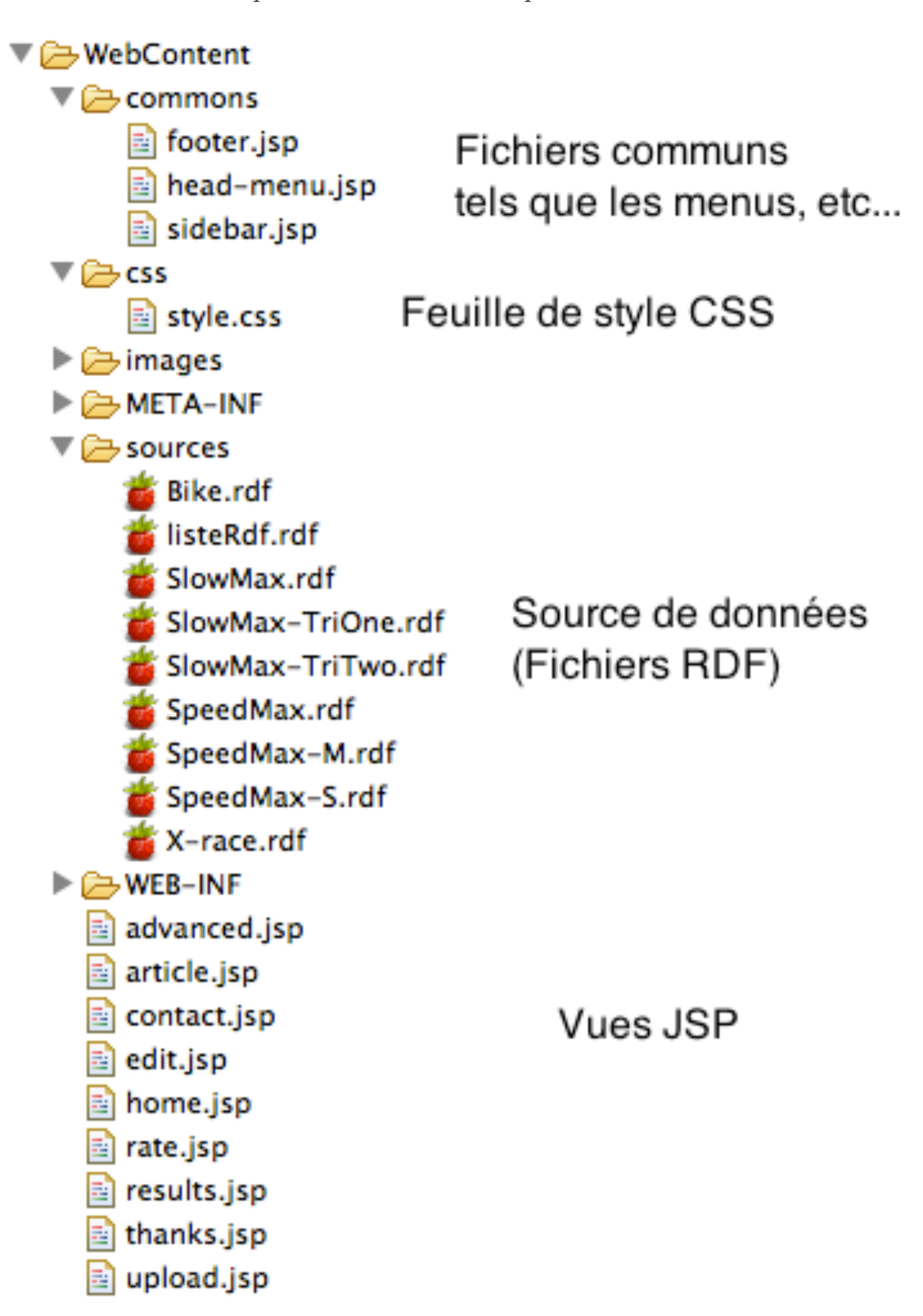
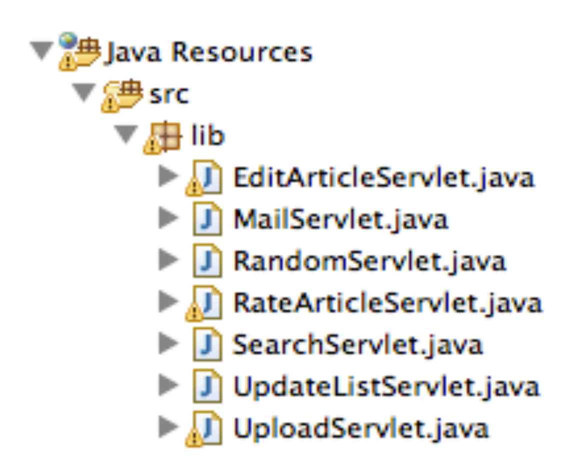


Figure 10 : Contenu du dossier WebContent

Contrôleurs

Les contrôleurs sont ici les différentes servlets créées au sein du système afin de vérifier l'intégrité des données saisies en cas de modification, de charger le contenu des pages JSP afin d'afficher les articles ou encore de rediriger les différentes exceptions générées.



EditArticleServlet est la servlet permettant la modification d'un article.

MailServlet sert à envoyer un mail via le formulaire contact dans le menu de l'application.

RandomServlet permet d'afficher un article au hasard.

RateArticleServlet permet de noter un article.

SearchServlet retourne la liste des articles trouvés en fonction du paramètre de recherche passé via la fonction de recherche de la classe JenaInit.

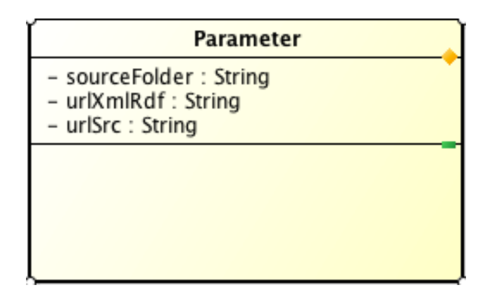
UpdateListServlet permet de mettre à jour le fichier listRDF en cas d'ajout de données dans la base. Ainsi, sans avoir une quelconque connaissance en XML/RDF, il est alors aisé de mettre à jour cette base.

UploadServlet permettant l'upload au sein de l'application de nouveaux fichiers RDF et faisant ensuite appel à la servlet UpdateListServlet afin de mettre à jour la liste des données.

Compléments

Une classe intermédiaire a également été créée. En effet, pour les besoins de l'application, nous avons créé une classe parameter permettant de géréer de manière indépendante au reste du code source les variables du type url et chemin d'accès aux fichiers.

En effet, il est dès lors plus aisé pour l'utilisateur désirant déployer l'application (cf. Annexes) de paramétrer les chemins requis.



c. Fonction de recherche – Jena

La fonction de recherche est réalisée avec l'API Jena, permettant d'exploiter aisément des fichiers RDF. Le fonctionnement se divise en deux parties, une partie initialisation et une partie recherche. Le code complet se situe dans une classe nommé ''JenaInit''.

La partie initialisation a pour but de créer un système de base de données contenant l'ensemble des fichiers RDF voulus. Pour ce faire une fonction prend en entrée l'adresse du fichier ''listeRdf.rdf'' qui regroupe l'ensemble des adresses des fichier RDF de la base de données. Cette fonction télécharge ensuite tous les fichiers RDF définis dans le fichier ''listeRdf'' pour créer le système de base de données complet. Ce qui permet de modifier la base de données des fichiers RDF sans modifier le code de l'application.

Pseudo code « initialisation » :

```
-Créer une base de données vide
-Télécharger le fichier ''listeRdf''
Pour chaque adresse de fichier RDF dans ''listeRdf''
        -Télécharger le fichier RDF
        -Ajouter dans la base de données le fichier RDF
Fin Pour
```

La partie recherche utilise le moteur ARQL de Jena pour exécuter les requêtes de recherche. Nous avons réalisé trois configurations différentes de recherche. La fonction prend donc en entré deux paramètres, le premier paramètre correspond au mode de recherche choisi et le deuxième à une suite de ''char'' faisant office de mot clé à rechercher. La fonction renvoie alors une liste de fichiers RDF classés par pertinence.

Le mode 1 exécute une recherche mot à mot sur le nœud Commentaire de tous les fichiers RDF. Il est réalisé grâce à une seule requête ARQL sur le système de donnée.

Le mode 2 exécute une recherche sur le nom d'un fichier RDF et sur ses héritages. Le but est de remonter l'arbre complet des héritages. Il est réalisé en combinent 2 listes, la première liste correspond au fichier RDF pertinent et traité et la deuxième liste correspond au fichier RDF pertinent non traité. L'algorithme correspond à parcourir en boucle la deuxième liste en mettent à jour la première jusqu'à ce que la deuxième soit vide.

Pseudo code ''recherche par nom et héritage'' :

```
-Créer une nouvelle liste nommée liste1 de RDF pertinent
-Créer une nouvelle liste nommée liste2 de RDF à traiter
-Exécution d'une requête ARQL qui recherche des noms de RDF pertinents en rapport avec le texte
passé en entrée et stoker le résultat dans liste1.
-Stoker dans liste2 les parents du résultat de la requête.
Tant que liste2 n'est pas vide faire
        -Exécution d'une requête ARQL qui recherche des noms de RDF pertinent en rapport avec le
        premier élément de la liste2 et stoker le résultat dans liste1.
        -Stoker dans liste2 les parents du résultat de la requête.
        -Supprimer le premier élément de la liste 2.
        -Supprimer les doublons dans liste1
        -Supprimer les doublons dans liste2
Fin Tant Que
```

Le mode 3 combine le mode 1 et 2 en ajoutent dans la liste des RDF à traiter tous les fichiers RDF trouvés par la méthode 1. Ce mode est un bon compromis entre les deux premiers modes car il permet d'avoir des résultats pertinents tous en ayant un plus grand choix de fichiers RDF affichés.

d. Fonctionnement de la classe Article

La classe Article possède 12 propriétés privées correspondant à chacune des propriétés d'un fichier RDF.

Nous disposons en effet des propriétés suivantes :

Propriété	Signification
Name	Nom (ou identifiant) du RDF
Role	Rôle du RDF
Creator	Auteur du fichier
Datedeb	Date de création du fichier
Datefin	Date de dernière modification du fichier
Maturite	Indice de maturité
Evaluation	Pourcentage d'évaluation
Commentaire	Commentaire de l'article
Other	Correspond à la totalité des propriétés facultatives susceptibles de changer en fonction de l'ontologie de l'article
Rate	Note globale de l'article
Parent	Identifiant (ou name) du fichier RDF parent
Nbraters	Nombre total de votants

Mis à part les différents accesseurs de ces propriétés (getters et setters), cette classe possède les différentes méthodes suivantes :

public Article(String title)	Constructeur de la classe Article, prenant en paramètre le nom du fichier RDF concerné.
Création du modèle factory à l'aide de la classe jenaInit. Exécution d'une requête ARQL permettant de récupérer toutes les informations du fichier Récupération de toutes les propriétés (name, …)	

public String displayStars()	Fonction permettant d'afficher les étoiles correspondant à la notation globale de l'article
Note = This.rate / This.nbRaters Tant que Note Afficher étoiles Fin Tant que	

public void saveRDF()	Méthode servant à sauvegarder les données mises à jour des RDF
Template de base avec les balises inhérentes à tout fichier Ajout de chaque propriété au fichier à l'aide des getters et setters Déclaration d'un filewriter pour écrire dans le fichier Ecriture (ou création) dans le fichier	

private String getOtherInfoRDF(InputStream)	**Fonction récupérant toutes les other info sans traitement préalable du fichier RDF.**
Tant que !EOF Si les propriétés ne sont pas celles de bases chaineRetour = ligne du fichier Fin Si Fin Tant que Retourne chaineRetour	

private String trtLine(String line)	**Fonction effectuant du traitement de String afin de récupérer uniquement le nom de la propriété (différent de la valeur)**
Ligne similaire à : <CyclingOnto:WheelName>Zipp 808 AR</CyclingOnto:WheelName> Récupération du nom en extrayant les infos après <CyclingOnto: Stop une fois que la balise se referme (>) Dasn cet exemple le nom de la propriété retourné sera WheelName	

private String trtTag(String ligne)	**Similaire à la fonction ci-dessus. Permet de récupérer cette fois la valeur de la propriété**
Ligne similaire à : <CyclingOnto:WheelName>Zipp 808 AR</CyclingOnto:WheelName> Récupération du nom en extrayant les infos après la fermeture de la première balise (>) Stop une fois qu'une la balise se referme (</) Dans cet exemple la valeur retournée sera Zipp 808 AR	

private String loadOtherInfoEdit(InputStream ips)	**Chargement des Other infos dans un formulaire**
Parcours du fichier RDF Pour chaque ligne du fichier Utilisation des deux fonctions ci-dessus pour récupérer les valeur et noms de propriétés Insertion dans des tags de type <input> pour le formulaire Fin Pour Fin parcours	

private String loadOtherInfo(InputStream ips)	**Chargement des Other infos en mode textuel cette fois-ci**
Parcours du fichier RDF Pour chaque ligne du fichier Utilisation des deux fonctions ci-dessus pour récupérer les valeur et noms de propriétés Insertion dans des tags de type <p> pour afficher les données en mode textuel Fin Pour Fin parcours	

e. Utilisation dans l'application

SearchServlet.java : Servlet effectuant une recherche au sein des fichiers RDF.

- Utilisation de la classe ''JenaInit''
- Définition du modèle
- Exécution de la fonction ''search'' retournant les résultats de la recherche
- Redirection du résultat dans la JSP ''results.jsp'' afin d'afficher le résultat
    - Pour chaque élément du tableau de résultats retourné
        - Appel du constructeur prenant en paramètre le nom du fichier
        - Accès à chacune des propriétés à l'aide des accesseurs getters et setters
        - Affichage des infos
    - Fin Pour

RandomServlet.java : Chargement d'un article au hasard.

- Récupération de la liste complète des fichiers RDF
- On en choisit un au hasard
- On redirige vers la JSP ''article.jsp'' qui va l'exploiter à l'aide de la classe article.
    - Appel du constructeur prenant en paramètre le nom du fichier
    - Accès à chacune des propriétés à l'aide des accesseurs getters et setters
    - Affichage des infos

UploadServlet.java : Servlet permettant l'upload d'un nouveau fichier RDF et mettant à jour le fichier listeRDF.rdf contenant la liste des adresses de chaque fichier.

- Upload du fichier avec un DiskFileUpload
- Mise à jour de ''listeRDF.rdf'' à l'aide de l'appel à la servlet UpdateListServlet
    - Insertion des balises de bases de type XML
    - Tant que des fichiers RDF existent
        - Ajout d'une nouvelle ligne dans le fichier
    - Fin Tant que

EditArticleServlet.java : Permet de mettre à jour un fichier RDF en fonction des modifications effectuées par l'utilisateur.

- Récupération de chaque info saisie par l'utilisateur dans le formulaire
- Déclaration d'une nouvelle instance de classe Article
- Utilisation des setters de la classe pour y affecter les valeurs du formulaire préalablement récupérées
- Appel de la méthode SaveRDF de la classe Article pour sauvegarder le fichier
- Redirection vers le descriptif de l'article pour constater les modifications

RateArticleServlet.java : Permet de mettre à jour la notation de l'article, à savoir les propriétés nbRaters et rate.

- Utilisation du constructeur prenant en paramètre le nom du fichier de la classe Article
- Utilisation du getter pour récupérer les valeurs du nombre de votant et de la note globale
- Incrémentation de + 1 pour le nombre de votant
- Incrémentation de + "note" provenant du formulaire de notation
- Utilisation des setters pour mettre à jour les données
- Appel à la fonction saveRDF pour mettre à jour les données

III. EVOLUTIONS POSSIBLES

a. Evolutivité de la fonction recherche

Système client-serveur

Le principal problème de notre implémentation de la fonction de recherche est la fonction initialisation qui est exécutée pour chaque recherche. Le temps d'exécution de cette fonction dépend fortement de la taille de la base de données. La création d'un serveur pour la fonction de recherche permettrait de ne faire l'initialisation qu'une seule fois au démarrage, ce qui diminuerait l'influence de la taille de la base de donnée pour la recherche.

Doublement de la base de données

Les paramètres de recherche ne dépendent pas de tous les nœuds des fichier RDF, il serait alors intéressent de créer un clone de notre base de donnée en ne prennent on compte uniquement les paramètre associé à la recherche, ce qui permettrait de diminuer la mémoire RAM utilisé sur le serveur pour la fonction de recherche.

b. Ajout de nouvelles fonctionnalités

Ci après, une liste non exhaustive de fonctionnalités utiles pouvant être incluses au sein de l'application afin de la rendre plus simple d'utilisation ou tout simplement pour permettre à l'utilisateur lambda d'aller encore plus loin.

✓ Création manuelle de nouveaux articles

Une fonctionnalité majeure manquante serait la création manuelle de nouveaux articles au sein de l'application. Une telle fonctionnalité permettrait d'éviter des erreurs de type syntaxe RDF ou absence de données et apporterait à l'utilisateur une plus grande marge de manœuvre concernant la gestion des données.

✓ Gestion des utilisateurs

La gestion des utilisateurs via une base de données pourrait être un plus dans le sens où des droits d'utilisation de l'application pourraient être appliqués en fonction des permissions d'accès accordés. En effet, la modification d'un article pourrait ne pas être permise à tout le monde par exemple.

✓ Parseur RDF

Un parseur RDF permettrait dans notre cas d'apporter une plus grande stabilité dans notre application. En effet, si un fichier RDF mal formé venait à être envoyé sur le serveur, plusieurs erreurs et exceptions seraient générées car l'application est très sensible aux erreurs RDF. Un tel module assurerait une plus grande stabilité et permettrait d'éviter toute erreur de la part de l'utilisateur.

Conclusion

Ce projet Wiki Sémantique nous a permis dans un premier temps d'appliquer les nombreuses méthodes abordées durant certaines de nos UV à l'UTBM, mais aussi de nous focaliser sur un autre aspect de la conception d'un projet, à savoir la recherche.

En effet, mis à part la partie Java/JEE, nos connaissances dans le domaine des fichiers RDF ainsi que la libraire Jena étaient quasiment nulles. De plus, avant d'arriver à la partie conception même du projet, il nous à fallu passer par des étapes intermédiaires telles que la recherche d'existant afin d'en exploiter les ressources.

C'est donc au terme de plusieurs semaines de recherches et d'exploration de pistes parfois infructueuses que nous avons mené à bien notre projet.

Annexes

Exemple d'un ficher RDF

Ci-après un exemple de fichier RDF bien formé avec respect de la syntaxe, de la fermeture correcte des balises, respect de la casse, …

```
<rdf:RDF
xmlns:CyclingOnto="http://KATRAS/AcivitePropriete/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
<rdf:Description rdf:about="http://utbm.acsp.fr/CP51_LO15_SlowMax Tri One/"> (1)
<CyclingOnto:NameRdf>SlowMax-TriOne</CyclingOnto:NameRdf> (2)
<CyclingOnto:RoleCreateurConnaissance>Concepteur</CyclingOnto:RoleCreateurConnais
sance>
<CyclingOnto:CreerPar>ILahoud</CyclingOnto:CreerPar>
<CyclingOnto:DateDebut>08/01/2009</CyclingOnto:DateDebut>
<CyclingOnto:DateFin>05/01/2012</CyclingOnto:DateFin>
<CyclingOnto:IndiceMaturite>10</CyclingOnto:IndiceMaturite>
<CyclingOnto:PourcentageEvaluation>25</CyclingOnto:PourcentageEvaluation>
<CyclingOnto:Commentaire> SlowMax CF 2010 TriOne</CyclingOnto:Commentaire>
<CyclingOnto:Rate>12</CyclingOnto:Rate>
<CyclingOnto:NbRaters>3</CyclingOnto:NbRaters>
<CyclingOnto:Parent>SlowMax</CyclingOnto:Parent>
<CyclingOnto:FrameWeight>1.700</CyclingOnto:FrameWeight>
<CyclingOnto:FramesetLength>XL</CyclingOnto:FramesetLength>
<CyclingOnto:SeatTubeLength>3600</CyclingOnto:SeatTubeLength>
<CyclingOnto:TopTubeLength>585</CyclingOnto:TopTubeLength>
<CyclingOnto:HeadTubeLength>175</CyclingOnto:HeadTubeLength>
<CyclingOnto:HeadTubeAngle>72</CyclingOnto:HeadTubeAngle>
<CyclingOnto:SeatTubeAngle>75</CyclingOnto:SeatTubeAngle>
<CyclingOnto:ChainstayLength>3900</CyclingOnto:ChainstayLength>
<CyclingOnto:WheelBase>1010</CyclingOnto:WheelBase>
<CyclingOnto:HandlebarName>Alu X</CyclingOnto:HandlebarName>
<CyclingOnto:HandlebarWeight>927</CyclingOnto:HandlebarWeight>
<CyclingOnto:HandlebarMaterial>Aluminuim</CyclingOnto:HandlebarMaterial>
<CyclingOnto:HandlebarLength>36</CyclingOnto:HandlebarLength>
<CyclingOnto:WheelName>Zipp 808 AR</CyclingOnto:WheelName>
<CyclingOnto:WheelWeight>808</CyclingOnto:WheelWeight>
<CyclingOnto:WheelMaterial>Carbone</CyclingOnto:WheelMaterial>
<CyclingOnto:WheelSize>26</CyclingOnto:WheelSize>
<CyclingOnto:cranksetName>Speed-One</CyclingOnto:cranksetName>
<CyclingOnto:cranksetWeight>112</CyclingOnto:cranksetWeight>
<CyclingOnto:cranksetMaterial>Alliage Alu-Carbone</CyclingOnto:cranksetMaterial>
<CyclingOnto:ChainRingTeeth>54</CyclingOnto:ChainRingTeeth>
</rdf:Description>
</rdf:RDF>
```

Règles de bonne formation :

1. rdf:about="http://utbm.acsp.fr/CP51_LO15_ + Nom Fichier RDF
2. Doit être identique au nom de fichier (case sensitive)
3. Les balises doivent être correectement fermées
   <ontologie:tag>Value</ontologie:tag>
4. Respect de l'architecture des balises
5. Le reste est identique au XML

Manuel d'installation

Avec ce manuel d'installation vous est fourni une archive contenant toutes les sources de l'application prête à être installée sur un ordinateur tiers afin de poursuivre le développement de l'outil.

Contenu de l'archive :

- Archive SemanticWiki.zip contenant le projet de l'application écrite en J2EE
- Archive jena.zip contenant les sources java utilisées pour effectuer les recherches au sein des RDF à l'aide de Jena.
- Archives libs.zip contenant les librairies Jena utilisées ainsi que les librairies pour la JSTL

Autres Informations Utiles :

- Eclipse
- Apache Tomcat V. 6.0 (http://tomcat.apache.org/download-60.cgi)
- JRE system Library : Java SE 6
    - Si librairie système différente, penser à la modifier dans le java build path…

Procédure d'installation :

L'installation de l'application s'effectue de la manière suivante :

1ère étape :

1. Créer un nouveau « java project » nommé jena.
2. Y extraire le contenu de l'archive jena.zip.
3. Ouvrir les propriétés du projet > java build path
4. Onglet libraires > add external jars
5. Sélectionner tous les jar de l'archive libs.zip

2ème étape :

1. Créer un nouveau « dynamic web application » nommée semanticWiki.
2. Target Runtime : Apache Tomcat v6.0 (si absent, télécharger tomcat V6.0. Puis, dans Eclipse, bouton « new runtime » Apache Tomcat V6.0. Next. Sélectionner le dossier d'installation de Tomcat. Finish.
3. Dynamic Web Module version : 2.5
4. Default configuration : default configuration for apache tomcat v6.0
5. Finish
6. Y extraire le contenu de l'archive semanticWiki.zip
7. Ouvrir les propriétés du projet > java build path
8. Onglet source > link Source
9. Sélectionner le dossier src du projet jena
10. Le renommer jena-src
11. Click finish + ok

3ème étape :

1. Dans le projet semanticWiki, ouvrir le dossier webcontent > web-inf > lib
2. Vérifier la présence des différentes librairies
3. Si aucune librairie, y ajouter le contenu de l'archive libs.zip
4. Ouvrir la classe Parameter présente dans le package param
5. Modifier les 3 paramètres pour que l'application les prenne en compte
    a. sourceFolder : Chemin relatif de l'endroit où sont stockées les données
    b. urlXmlRdf : URL indiquant le lieu où se situe le fichier listeRdf.rdf
    c. urlSrc : URL indiquanr où se situent les sources de données de l'application
    d. En règle générale, uniquement le paramètre sourceFolder doit être modifié

4ème étape :

1. Compiler le projet semanticWiki
    a. Click droit sur le projet > run as > run on server
    b. Manually define a new server
    c. Tomcat v6.0
    d. Finish
2. Ouvrir le navigateur web
3. Saisir l'url suivante : http://localhost:8080/semanticWiki/home.jsp
4. Pour effectuer une recherche, saisir le mot clé suivant : Speed (case sensitive) ou Bike…
5. Naviguer au travers du site

Vérification

Arborescence du workspace :

- Dossier Jena
    - bin
    - src
- Dossier semanticWiki
    - build
    - src
    - webcontent
- Dossier Servers
    - Tomcat v6.0 Server at localhost-config